FIT2099
Assignment 1
Design Rationale

Members:
Abhishek Shrestha
Amindu Kaushal

Objective: Explain how the system will work. Describe reasoning behind design decisions

Design Principles:

1. Avoid Excessive use of literals
2. Don't repeat yourself (inheritance)
3. Classes should be responsible for their own properties
4. Reduce dependencies as much as possible
5. Group elements that must depend on each other inside an encapsulation boundary
6. Minimize dependencies across encapsulation boundaries
7. Declare things in the tightest possible scope
8. Delegation over inheritance
9. Fail fast principle
10. Command query principle (method can change state or return state but not both)
11. Liskov design principle

Plan: Break rationale down into separate components (features / interactions)

Items that Populate the GameMap (Tree, Dirt, Bush)
In this section, we have included objects that will be present in the world as terrain. These include the Tree and Dirt classes from the game package. We have added a new class Bush, that inherits Ground (similar to Tree and Dirt). This allows us to use the existing functionality of the Ground class as such tick() and can allowableActions(). Similarly, we have inherited the PortableItem class to create a new Fruit class, as we fruits can be picked by the player. This is an example of the Don't Repeat yourself (DRY) principle.

Though the assignment specifications hint on the Tree and Bush class having association with the Fruit class, we have Reduced Dependencies between these classes by deciding to store the number of Fruits each Tree or Bush object will have as integers, instead of storing a collection of Fruit objects for each Tree or Bush. Whenever other Actors interact with the Trees or Bushes to search for Fruit or when fruits fall from trees, given that the interaction is successful, we plan on instantiating a new instance of Fruit at the location of the Tree or Bush. Since a Tree or Bush class only needs to deal with producing fruits, we have a growFruitAction, that simply increments the number of fruits in a bush or tree. This simply changes the state of the game, and nothing more, adhering to the command query principle.

TODO: growBushAction

Dinosaur Classes
In this section, we will talk about the three dinosaur classes Stegosaur, Brachiosaur and Allosaur. Since these classes would be very similar, we have made an abstract base class

called Dinosaur, that in turn inherits the Actor class from the engine. This allows us to store a lot of methods and properties that three dinosaurs would share such as food level, capability (carnivorous or herbivorous), mate behavior etc. in a single abstract class, following the Don't Repeat yourself (DRY) principle. Creating three different classes instead having a of a single Dinosaur class and creating multiple types of dinosaurs objects by specifying "Stegosaur", "Brachiosaur" or "Allosaur" avoids excessive use of literals in the code, and makes the code more structured.

Similarly, by having a separate class for Allosaur, we can deal with actions such as EatMeatAction without involving other dinosaur classes, following the principle Declare things in the tightest possible scope.

TODO: more